17장 생성자 함수에 의한 객체 생성

생성자 함수란?

new 연산자와 함께 호출하여 객체(인스턴스)를 생성하는 함수

객체 리터럴에 의한 객체생성의 문제점

객체는

- **프로퍼티**로 객체의 **상태**
- **메서드**로 객체의 **동작**

을 표현한다

단점

중복 작업 필요

```
const circle1 = {
    radius: 5,
    getDiameter(){
        return 2 * this.radius;
    }
};

console.log(circle1.getDiameter());

const circle2 = {
    radius: 10,
    getDiameter(){
        return 2 * this.radius;
    }
};
```

```
console.log(circle2.getDiameter());
```

프로퍼티 구조가 동일하더라고 매번 같은 프로퍼티와 메서드를 기술해야한다.

해결

생성자 함수를 사용하여 생성 시 간편하게 생성할 수 있다

```
// 생성자 함수
function Circle1(radius) {
    this.radius = radius;
    this.getDiameter = function () {
        return 2 * this.radius;
    };
}

// 인스턴스의 생성
const circle1 = new Circle(5);
const circle2 = new Circle(10);

console.log(Circle2.getDiameter());
console.log(Circle2.getDiameter());
```

this

객체 자신의 프로퍼티 나 메서드를 참조하기 위한 자기 참조 변수이다.

함수 호출 방식에 따라 동적으로 바인딩된다

호출 방식	this가 가리키는 값
일반함수	**전역객체**
메서드	**메서드를 호출한 객체**
생성자 함수	생성자함수가 생성할 **인스턴스**

생성자 함수의 역할

1. 인스턴스를 생성하기 위한 템플릿(클래스)로 동작하여 인스턴스를 생성
2. 생성된 인스턴스를 초기화 (옵션

```
function Circle1(radius) {
    // 1. 암묵적으로 인스턴스가 생성되고 this에 바인딩된다.
    console.log(this); // Circle {}

    this.radius = radius;
    this.getDiameter = function () {
        return 2 * this.radius;
    };
}
```

이 과정에서 엔진은 암묵적인 처리를 통해 인스턴스를 생성하고 반환한다.

(따로 반환하는 코드가 있지 않음)

생성자 함수의 인스턴스 생성과정

1. 인스턴스 생성
    - 암묵적으로 빈 객체 생성
2. this 바인딩
    - 생성된 빈 객체 즉 인스턴스 는 this에 바인딩 된다.
3. 인스턴스 초기화
    - this에 바인딩되어 있는 인스턴스를 초기화
    - 이 순서에서 개발자의 기술에 따라 고정값을 할당할 수 있다.
4. 인스턴스 반환
    - 완성된 인스턴스가 바인딩된 this에 암묵적으로 반환된다.
    - 주의
        - 생성자 함수에서 return 값을 준다면?
            1. 객체 return

- this가 반환되지 못하고 명시된 객체가 반환된다.

2. 원시값 return
    - 원시 값 반환은 무시되고 암묵적으로 this가 반환된다.

결론적으로 둘다 생성자 함수의 기본동작을 훼손하기 때문에

생성자 함수 내부에서의 return 문은 **반드시 생략** 해야한다.

함수의 구분

함수는 기본적으로 객체이지만 일반 객체 와 다르다.

그리하여 함수로 동작하기 위한

1. 내부 슬롯 [[Environment]], [[FormalParameters]]
2. 내부 메서드 [[Call]], [[Construct]]

를 추가로 가지고 있다

구분하자면

[[[Call]]를 갖는 함수객체는 callable이라 불리며 , 호출가능하다

[[Construct]] 를 갖는 함수객체는 constructor 이라 불리며 , 생성자 함수로써 호출가능하다

[[Construct]] 를 갖지 않는 함수객체는 non-constructor 이라 불리며 , 생성자 함수로써 호출가능하다

모든 함수는 [[[Call]] 을 갖고 있으므로 호출 가능하다

constructor 와 non-constructor의 구분

• constructor: 함수 선언문, 함수 표현식, 클래스
• non-constructor : 메서드, 화살표 함수

여기서 메서드는 ES6의 메서드 축약 표현 만을 의미한다

```
//ES6의 메서드 축약 표현
const mj = {
    x () {}
}
```

non-constructor는 [[Construct]]를 가지고 있지 않기 때문에

생성자 함수로서 호출시 에러가 발생한다.

new 연산자

new 연산자를 통해 함수를 호출하면 [[[Call]]이 아닌 [[Construct]]가 호출된다.

this

new 연산자를 통해 호출

    내부의 this는 생성자 함수가 생성할 인스턴스를 가리킨다.

일반함수로 호출

    내부의 this는 전역객체를 가리킨다

생성자함수 가 new 연산자 없이 호출방지

1. new.target

생성자함수 가 new 연산자 없이 호출되는것 을 방지하기 위해 사용

```
function Circle(radius){
    if(!new.target){
        return new Circle(radius);
    }
    this.radius = radius;
    this.getDiameter = function () {
        return 2 * this.radius;
    };
}
```

```
const circle = Circle(5); // new 연산자 없이 생성자 함수를 호출하여도
console.log(circle.getDiameter());
```

2. 스코프 세이프 생성자 패턴

instanceof 연산자 사용하여 처리

```
// Scope-Safe Constructor Pattern
function Circle(radius) {
  // 생성자 함수가 new 연산자와 함께 호출되면 함수의 선두에서 빈 객체를 생
  // this에 바인딩한다. 이때 this와 Circle은 프로토타입에 의해 연결된다

  // 이 함수가 new 연산자와 함께 호출되지 않았다면 이 시점의 this는 전역
  // 즉, this와 Circle은 프로토타입에 의해 연결되지 않는다.
  if (!(this instanceof Circle)) {
    // new 연산자와 함께 호출하여 생성된 인스턴스를 반환한다.
    return new Circle(radius);
  }

  this.radius = radius;
  this.getDiameter = function () {
    return 2 * this.radius;
  };
}

// new 연산자 없이 생성자 함수를 호출하여도 생성자 함수로서 호출된다.
const circle = Circle(5);
console.log(circle.getDiameter()); // 10
```

new 연산자 없이 생성자 함수를 호출하여도 new.target을 통해 생성자 함수로서 호출된다.

Reference

- https://hong-p.github.io/javascript/javascript-deepdive-ch17/

- https://velog.io/@049494/17장.-생성자-함수에-의한-객체-생성